MySQL Exercise 12: Queries that Test Relationships Between Test Completion and Testing Circumstances

In this lesson, we are going to practice integrating more of the concepts we learned over the past few weeks to address whether issues in our Dognition sPAP are related to the number of tests dogs complete. We are going to focus on a subset of the issues listed in the "Features of Testing Circumstances" branch of our sPAP. You will need to look up new functions several times and the final queries at which we will arrive by the end of this lesson will be quite complex, but we will work up to them step-by-step.

To begin, load the sql library and database, and make the Dognition database your default database:

In [5]:

```
%load_ext sql
%sql mysql://studentuser:studentpw@localhost/dognitiondb
%sql USE dognitiondb
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
 * mysql://studentuser:***@localhost/dognitiondb
0 rows affected.
```

Out[5]:

```
[]
```

In order to make it easier to practice SQL queries with meaningful examples before we learned how to join tables, I added extra columns to the "dogs" table that were not in the original Dognition database. These extra columns included the "total_tests_completed" field and multiple inter-test-interval ("iti") summary fields. **Please do NOT try to use these extra fields in the query exercises below. Since you now know how to join tables, we will practice writing queries as if you only had the data provided in the original Dognition database.**

1. During which weekdays do Dognition users complete the most tests?

The first question we are going to address is whether there is a certain day of the week when users are more or less likely to complete Dognition tests. If so, targeting promotions or reminder emails to those times of the week might increase the number of tests users complete.

At first, the query we need to address this question might seem a bit intimidating, but once you can describe what the query needs to do in words, writing the query won't seem so challenging.

Ultimately, we want a count of the number of tests completed on each day of the week, with all of the dog_guids and user_guids the Dognition team flagged in their exclude column excluded. To achieve this, we are going to have to use the GROUP BY clause to break up counts of the records in the completed_tests table according to days of the week. We will also have to join the completed_tests table with the dogs and users table in order to exclude completed_tests records that are associated with dog_guids or user_guids that should be excluded. First, though, we need a method for extracting the day of the week from a time stamp. In MySQL Exercise 2 we used a function called "DAYNAME". That is the most efficient function to use for this purpose, but not all database systems have this function, so let's try using a different method for the queries in this lesson. Search these sites to find a function that will output a number from 1-7 for time stamps where 1 = Sunday, 2 = Monday, …, 7 = Saturday:

https://dev.mysql.com/doc/refman/5.7/en/sql-function-reference.html (https://dev.mysql.com/doc/refman/5.7/en/sql-function-reference.html)
http://www.w3resource.com/mysql/mysql-functions-and-operators.php (http://www.w3resource.com/mysql/mysql-functions-and-operators.php)

Question 1: Using the function you found in the websites above, write a query that will output one column with the original created_at time stamp from each row in the completed_tests table, and another column with a number that represents the day of the week associated with each of those time stamps. Limit your output to 200 rows starting at row 50.

In [6]:

```
%%sql
SELECT created_at, DAYOFWEEK(created_at)
FROM complete_tests
LIMIT 49,20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[6]:

created_at	DAYOFWEEK(created_at)
2013-02-05 22:10:06	3
2013-02-05 22:23:49	3
2013-02-05 22:26:36	3
2013-02-05 22:29:02	3
2013-02-05 22:32:25	3
2013-02-05 22:33:09	3
2013-02-05 22:36:11	3
2013-02-05 22:38:01	3
2013-02-05 22:48:58	3
2013-02-05 22:53:45	3
2013-02-05 22:59:45	3
2013-02-05 23:01:38	3
2013-02-05 23:04:43	3
2013-02-05 23:06:10	3
2013-02-05 23:35:48	3
2013-02-05 23:40:57	3
2013-02-05 23:45:30	3
2013-02-05 23:48:46	3
2013-02-05 23:54:40	3
2013-02-05 23:59:15	3

Of course, the results of the query in Question 1 would be much easier to interpret if the output included the name of the day of the week (or a relevant abbreviation) associated with each time stamp rather than a number index.

Question 2: Include a CASE statement in the query you wrote in Question 1 to output a third column that provides the weekday name (or an appropriate abbreviation) associated with each created_at time stamp.

In [7]:

```
%%sql
SELECT created_at, DAYOFWEEK(created_at),
       (CASE
        WHEN DAYOFWEEK(created_at)=1 THEN "Su"
        WHEN DAYOFWEEK(created_at)=2 THEN "Mo"
        WHEN DAYOFWEEK(created_at)=3 THEN "Tu"
        WHEN DAYOFWEEK(created_at)=4 THEN "We"
        WHEN DAYOFWEEK(created_at)=5 THEN "Th"
        WHEN DAYOFWEEK(created_at)=6 THEN "Fr"
        WHEN DAYOFWEEK(created_at)=7 THEN "Sa"
        END) AS daylabel
FROM complete_tests
LIMIT 49,20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[7]:

created_at	DAYOFWEEK(created_at)	daylabel
2013-02-05 22:10:06	3	Tu
2013-02-05 22:23:49	3	Tu
2013-02-05 22:26:36	3	Tu
2013-02-05 22:29:02	3	Tu
2013-02-05 22:32:25	3	Tu
2013-02-05 22:33:09	3	Tu
2013-02-05 22:36:11	3	Tu
2013-02-05 22:38:01	3	Tu
2013-02-05 22:48:58	3	Tu
2013-02-05 22:53:45	3	Tu
2013-02-05 22:59:45	3	Tu
2013-02-05 23:01:38	3	Tu
2013-02-05 23:04:43	3	Tu
2013-02-05 23:06:10	3	Tu
2013-02-05 23:35:48	3	Tu
2013-02-05 23:40:57	3	Tu
2013-02-05 23:45:30	3	Tu
2013-02-05 23:48:46	3	Tu
2013-02-05 23:54:40	3	Tu
2013-02-05 23:59:15	3	Tu

Now that we are confident we have the correct syntax for extracting weekday labels from the created_at time stamps, we can start building our larger query that examines the number of tests completed on each weekday.

Question 3: Adapt the query you wrote in Question 2 to report the total number of tests completed on each weekday. Sort the results by the total number of tests completed in descending order. You should get a total of 33,190 tests in the Sunday row of your output.

In [8]:

```
%%sql
SELECT DAYOFWEEK(created_at),COUNT(created_at) AS numtests,
       (CASE
        WHEN DAYOFWEEK(created_at)=1 THEN "Su"
        WHEN DAYOFWEEK(created_at)=2 THEN "Mo"
        WHEN DAYOFWEEK(created_at)=3 THEN "Tu"
        WHEN DAYOFWEEK(created_at)=4 THEN "We"
        WHEN DAYOFWEEK(created_at)=5 THEN "Th"
        WHEN DAYOFWEEK(created_at)=6 THEN "Fr"
        WHEN DAYOFWEEK(created_at)=7 THEN "Sa"
        END) AS daylabel
FROM complete_tests
GROUP BY daylabel
ORDER BY numtests DESC;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
7 rows affected.
```

Out[8]:

DAYOFWEEK(created_at)	numtests	daylabel
1	33190	Su
2	30195	Mo
3	27989	Tu
7	27899	Sa
4	26473	We
5	24420	Th
6	23080	Fr

So far these results suggest that users complete the most tests on Sunday night and the fewest tests on Friday night. We need to determine if this trend remains after flagged dog_guids and user_guids are excluded. Let's start by removing the dog_guids that have an exclude flag. We'll exclude user_guids with an exclude flag in later queries.

Question 4: Rewrite the query in Question 3 to exclude the dog_guids that have a value of "1" in the exclude column (Hint: this query will require a join.) This time you should get a total of 31,092 tests in the Sunday row of your output.

In [9]:

```
%%sql
SELECT DAYOFWEEK(c.created_at),COUNT(c.created_at) AS numtests,
        (CASE
         WHEN DAYOFWEEK(c.created_at)=1 THEN "Su"
         WHEN DAYOFWEEK(c.created_at)=2 THEN "Mo"
         WHEN DAYOFWEEK(c.created_at)=3 THEN "Tu"
         WHEN DAYOFWEEK(c.created_at)=4 THEN "We"
         WHEN DAYOFWEEK(c.created_at)=5 THEN "Th"
         WHEN DAYOFWEEK(c.created_at)=6 THEN "Fr"
         WHEN DAYOFWEEK(c.created_at)=7 THEN "Sa"
         END) AS daylabel
FROM complete_tests c JOIN dogs d
  ON c.dog_guid=d.dog_guid
WHERE d.exclude IS NULL OR d.exclude=0
GROUP BY daylabel
ORDER BY numtests DESC;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
7 rows affected.
```

Out[9]:

DAYOFWEEK(c.created_at)	numtests	daylabel
1	31092	Su
2	28250	Mo
7	26231	Sa
3	25764	Tu
4	24501	We
5	22347	Th
6	21028	Fr

Now we need to exclude the user_guids that have a value of "1" in the exclude column as well. One way to do this would be to join the completed_tests, dogs, and users table with a sequence of inner joins. However, we've seen in previous lessons that there are duplicate rows in the users table. These duplicates will get passed through the join and will affect the count calculations. To illustrate this, compare the following two queries.

Question 5: Write a query to retrieve all the dog_guids for users common to the dogs and users table using the traditional inner join syntax (your output will have 950,331 rows).

CODE

%%sql SELECT dog_guid FROM dogs d INNER JOIN users u ON d.user_guid=u.user_guid;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 950331 rows affected.

In [11]:

```
%%sql
SELECT dog_guid
FROM dogs d INNER JOIN users u
ON d.user_guid=u.user_guid
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[11]:

dog_guid
fd27b272-7144-11e5-ba71-058fbc01cf0b
fd417cac-7144-11e5-ba71-058fbc01cf0b
fd27b5ba-7144-11e5-ba71-058fbc01cf0b
fd3fb0f2-7144-11e5-ba71-058fbc01cf0b
fd27b6b4-7144-11e5-ba71-058fbc01cf0b
fd27b79a-7144-11e5-ba71-058fbc01cf0b
fd27b86c-7144-11e5-ba71-058fbc01cf0b
fd27ba1a-7144-11e5-ba71-058fbc01cf0b
fd27e9a4-7144-11e5-ba71-058fbc01cf0b
fd27ed46-7144-11e5-ba71-058fbc01cf0b
fd3cf718-7144-11e5-ba71-058fbc01cf0b
fd3d587a-7144-11e5-ba71-058fbc01cf0b
fd3fbfe8-7144-11e5-ba71-058fbc01cf0b
fd41c400-7144-11e5-ba71-058fbc01cf0b
fd42e33a-7144-11e5-ba71-058fbc01cf0b
fd3cffe2-7144-11e5-ba71-058fbc01cf0b
fd42e196-7144-11e5-ba71-058fbc01cf0b
fd453b6c-7144-11e5-ba71-058fbc01cf0b
fd43c0c0-7144-11e5-ba71-058fbc01cf0b
fd27b948-7144-11e5-ba71-058fbc01cf0b

Question 6: Write a query to retrieve all the *distinct* dog_guids common to the dogs and users table using the traditional inner join syntax (your output will have 35,048 rows).

CODE

%%sql SELECT DISTINCT dog_guid FROM dogs d JOIN users u ON d.user_guid=u.user_guid;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 35048 rows affected.

In [13]:

```
%%sql
SELECT DISTINCT dog_guid
FROM dogs d JOIN users u
  ON d.user_guid=u.user_guid
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[13]:

dog_guid
fd27b272-7144-11e5-ba71-058fbc01cf0b
fd417cac-7144-11e5-ba71-058fbc01cf0b
fd27b5ba-7144-11e5-ba71-058fbc01cf0b
fd3fb0f2-7144-11e5-ba71-058fbc01cf0b
fd27b6b4-7144-11e5-ba71-058fbc01cf0b
fd27b79a-7144-11e5-ba71-058fbc01cf0b
fd27b86c-7144-11e5-ba71-058fbc01cf0b
fd27ba1a-7144-11e5-ba71-058fbc01cf0b
fd27e9a4-7144-11e5-ba71-058fbc01cf0b
fd27ed46-7144-11e5-ba71-058fbc01cf0b
fd3cf718-7144-11e5-ba71-058fbc01cf0b
fd3d587a-7144-11e5-ba71-058fbc01cf0b
fd3fbfe8-7144-11e5-ba71-058fbc01cf0b
fd41c400-7144-11e5-ba71-058fbc01cf0b
fd42e33a-7144-11e5-ba71-058fbc01cf0b
fd3cffe2-7144-11e5-ba71-058fbc01cf0b
fd42e196-7144-11e5-ba71-058fbc01cf0b
fd453b6c-7144-11e5-ba71-058fbc01cf0b
fd43c0c0-7144-11e5-ba71-058fbc01cf0b
fd27b948-7144-11e5-ba71-058fbc01cf0b

The strategy we will use to handle duplicate rows in the users table will be to, first, write a subquery that retrieves the distinct dog_guids from an inner join between the dogs and users table with the appropriate records excluded. Then, second, we will join the result of this subquery to the complete_tests table and group the results according to the day of the week.

Question 7: Start by writing a query that retrieves distinct dog_guids common to the dogs and users table, excuding dog_guids and user_guids with a "1" in their respective exclude columns (your output will have 34,121 rows).

In [14]:

```
%%sql
SELECT DISTINCT dog_guid
FROM dogs d JOIN users u
ON d.user_guid=u.user_guid
WHERE (u.exclude IS NULL OR u.exclude=0) AND (d.exclude IS NULL OR
d.exclude=0)
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[14]:

dog_guid
fd27b272-7144-11e5-ba71-058fbc01cf0b
fd417cac-7144-11e5-ba71-058fbc01cf0b
fd27b5ba-7144-11e5-ba71-058fbc01cf0b
fd3fb0f2-7144-11e5-ba71-058fbc01cf0b
fd27b6b4-7144-11e5-ba71-058fbc01cf0b
fd27b79a-7144-11e5-ba71-058fbc01cf0b
fd27b948-7144-11e5-ba71-058fbc01cf0b
fd27bbbe-7144-11e5-ba71-058fbc01cf0b
fd27c1c2-7144-11e5-ba71-058fbc01cf0b
fd27c0fa-7144-11e5-ba71-058fbc01cf0b
fd27c5be-7144-11e5-ba71-058fbc01cf0b
fd27c74e-7144-11e5-ba71-058fbc01cf0b
fd27c64a-7144-11e5-ba71-058fbc01cf0b
fd27c7d0-7144-11e5-ba71-058fbc01cf0b
fd27c852-7144-11e5-ba71-058fbc01cf0b
fd27c8d4-7144-11e5-ba71-058fbc01cf0b
fd27cd98-7144-11e5-ba71-058fbc01cf0b
fd27ce1a-7144-11e5-ba71-058fbc01cf0b
fd3d249a-7144-11e5-ba71-058fbc01cf0b
fd27c956-7144-11e5-ba71-058fbc01cf0b

Question 8: Now adapt your query from Question 4 so that it inner joins on the result of the subquery you wrote in Question 7 instead of the dogs table. This will give you a count of the number of tests completed on each day of the week, excluding all of the dog_guids and user_guids that the Dognition team flagged in the exclude columns.

In [15]:

```
%%sql
SELECT DAYOFWEEK(c.created_at) AS dayasnum, YEAR(c.created_at) AS year, COUNT(c.created_at) AS numtests,
         (CASE
          WHEN DAYOFWEEK(c.created_at)=1 THEN "Su"
          WHEN DAYOFWEEK(c.created_at)=2 THEN "Mo"
          WHEN DAYOFWEEK(c.created_at)=3 THEN "Tu"
          WHEN DAYOFWEEK(c.created_at)=4 THEN "We"
          WHEN DAYOFWEEK(c.created_at)=5 THEN "Th"
          WHEN DAYOFWEEK(c.created_at)=6 THEN "Fr"
          WHEN DAYOFWEEK(c.created_at)=7 THEN "Sa"
          END) AS daylabel
FROM complete_tests c JOIN (SELECT DISTINCT dog_guid
                            FROM dogs d JOIN users u
                             ON d.user_guid=u.user_guid
                            WHERE ((u.exclude IS NULL OR u.exclude=0)
                            AND (d.exclude IS NULL OR d.exclude=0))) AS dogs_cleaned
  ON c.dog_guid=dogs_cleaned.dog_guid
GROUP BY daylabel
ORDER BY numtests DESC;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
7 rows affected.
```

Out[15]:

dayasnum	year	numtests	daylabel
1	2013	31036	Su
2	2013	28138	Mo
7	2013	26149	Sa
3	2013	25696	Tu
4	2013	24433	We
5	2014	22323	Th
6	2013	21027	Fr

These results still suggest that Sunday is the day when the most tests are completed and Friday is the day when the fewest tests are completed. However, our first query suggested that more tests were completed on Tuesday than Saturday; our current query suggests that slightly more tests are completed on Saturday than Tuesday, now that flagged dog_guids and user_guids are excluded.

It's always a good idea to see if a data pattern replicates before you interpret it too strongly. The ideal way to do this would be to have a completely separate and independent data set to analyze. We don't have such a data set, but we can assess the reliability of the day of the week patterns in a different way. We can test whether the day of the week patterns are the same in all years of our data set.

Question 9: Adapt your query from Question 8 to provide a count of the number of tests completed on each weekday of each year in the Dognition data set. Exclude all dog_guids and user_guids with a value of "1" in their exclude columns. Sort the output by year in ascending order, and then by the total number of tests completed in descending order. HINT: you will need a function described in one of these references to retrieve the year of each time stamp in the created_at field:

https://dev.mysql.com/doc/refman/5.7/en/sql-function-reference.html (https://dev.mysql.com/doc/refman/5.7/en/sql-function-reference.html)
http://www.w3resource.com/mysql/mysql-functions-and-operators.php (http://www.w3resource.com/mysql/mysql-functions-and-operators.php)

In [16]:

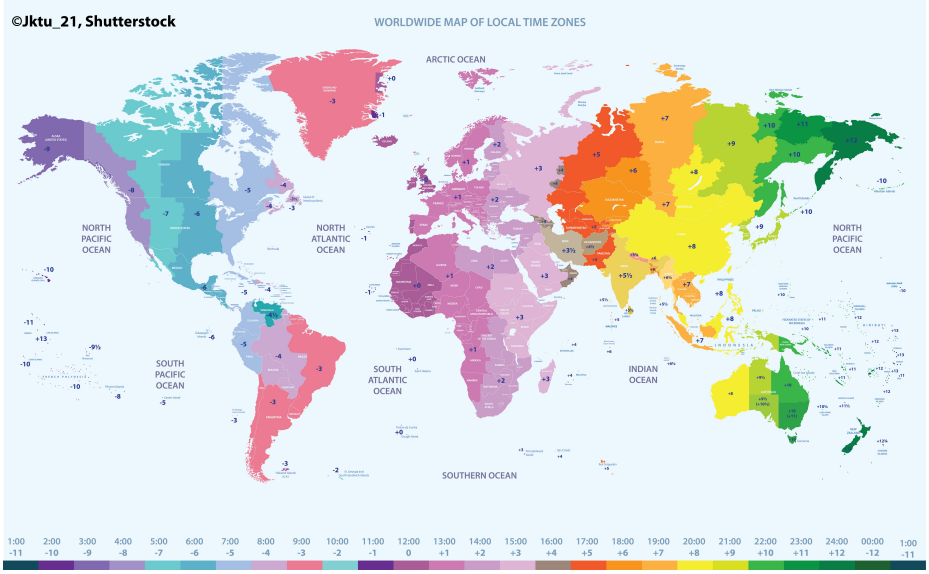
```
%%sql SELECT DAYOFWEEK(c.created_at) AS dayasnum, YEAR(c.created_at) AS year, COUNT(c.created_at) AS numtests,
                  (CASE
                   WHEN DAYOFWEEK(c.created_at)=1 THEN "Su"
                   WHEN DAYOFWEEK(c.created_at)=2 THEN "Mo"
                   WHEN DAYOFWEEK(c.created_at)=3 THEN "Tu"
                   WHEN DAYOFWEEK(c.created_at)=4 THEN "We"
                   WHEN DAYOFWEEK(c.created_at)=5 THEN "Th"
                   WHEN DAYOFWEEK(c.created_at)=6 THEN "Fr"
                   WHEN DAYOFWEEK(c.created_at)=7 THEN "Sa"
                   END) AS daylabel
FROM complete_tests c JOIN (SELECT DISTINCT dog_guid
                            FROM dogs d JOIN users u
                              ON d.user_guid=u.user_guid
                            WHERE ((u.exclude IS NULL OR u.exclude=0)
                            AND (d.exclude IS NULL OR d.exclude=0))) AS dogs_cleaned
  ON c.dog_guid=dogs_cleaned.dog_guid
GROUP BY year,daylabel
ORDER BY year ASC, numtests DESC;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
21 rows affected.
```

Out[16]:

dayasnum	year	numtests	daylabel
1	2013	8203	Su
7	2013	6854	Sa
2	2013	5740	Mo
4	2013	5665	We
3	2013	5393	Tu
6	2013	4997	Fr
5	2013	4961	Th
2	2014	9309	Mo
1	2014	9210	Su
3	2014	9177	Tu
4	2014	8857	We
7	2014	8257	Sa
5	2014	7286	Th
6	2014	6475	Fr
1	2015	13623	Su
2	2015	13089	Mo
3	2015	11126	Tu
7	2015	11038	Sa
5	2015	10076	Th
4	2015	9911	We
6	2015	9555	Fr

These results suggest that although the precise order of the weekdays with the most to fewest completed tests changes slightly from year to year, Sundays always have a lot of completed tests, and Fridays always have the fewest or close to the fewest completed tests. So far, it seems like it might be a good idea for Dognition to target reminder or encouragement messages to customers on Sundays. However, there is one more issue our analysis does not address. All of the time stamps in the created_at column are in Coordinated Universal Time (abbreviated UTC). This is a time convention that is constant around the globe. Nonetheless, as the picture below illustrates, countries and states have different time zones. The same UTC time can correspond with local times in different countries that are as much as 24 hours apart:



Therefore, the weekdays we have extracted so far may not accurately reflect the weekdays in the local times of different countries. The only way to correct the time stamps for time zone differences is to obtain a table with the time zones of every city, state, or country. Such a table was not available to us in this course, but we can run some analyses that approximate a time zone correction for United States customers.

Question 10: First, adapt your query from Question 9 so that you only examine customers located in the United States, with Hawaii and Alaska residents excluded. HINTS: In this data set, the abbreviation for the United States is "US", the abbreviation for Hawaii is "HI" and the abbreviation for Alaska is "AK". You should have 5,860 tests completed on Sunday of 2013.

In [17]:

```
%%sql
SELECT DAYOFWEEK(c.created_at) AS dayasnum, YEAR(c.created_at) AS year, COUNT(c.created_at) AS numtests,
              (CASE
               WHEN DAYOFWEEK(c.created_at)=1 THEN "Su"
               WHEN DAYOFWEEK(c.created_at)=2 THEN "Mo"
               WHEN DAYOFWEEK(c.created_at)=3 THEN "Tu"
               WHEN DAYOFWEEK(c.created_at)=4 THEN "We"
               WHEN DAYOFWEEK(c.created_at)=5 THEN "Th"
               WHEN DAYOFWEEK(c.created_at)=6 THEN "Fr"
               WHEN DAYOFWEEK(c.created_at)=7 THEN "Sa"
               END) AS daylabel
FROM complete_tests c JOIN
                       (SELECT DISTINCT dog_guid
                        FROM dogs d JOIN users u
                        ON d.user_guid=u.user_guid
                        WHERE ((u.exclude IS NULL OR u.exclude=0)
                        AND u.country="US"
                        AND (u.state!="HI" AND u.state!="AK")
                        AND (d.exclude IS NULL OR d.exclude=0))) AS dogs_cleaned
    ON c.dog_guid=dogs_cleaned.dog_guid
GROUP BY year,daylabel
ORDER BY year ASC, numtests DESC;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
21 rows affected.
```

Out[17]:

dayasnum	year	numtests	daylabel
1	2013	5860	Su
7	2013	4674	Sa
2	2013	3695	Mo
4	2013	3496	We
3	2013	3449	Tu
6	2013	3163	Fr
5	2013	3090	Th
2	2014	7278	Mo
3	2014	6800	Tu
1	2014	6632	Su
4	2014	6331	We
7	2014	6006	Sa
5	2014	5271	Th
6	2014	4831	Fr
2	2015	8784	Mo
1	2015	8570	Su
3	2015	7218	Tu
7	2015	7116	Sa
5	2015	6343	Th
4	2015	6164	We
6	2015	5689	Fr

The next step is to adjust the created_at times for differences in time zone. Most United States states (excluding Hawaii and Alaska) have a time zone of UTC time -5 hours (in the eastern-most regions) to -8 hours (in the western-most regions). To get a general idea for how much our weekday analysis is likely to change based on time zone, we will subtract 6 hours from every time stamp in the complete_tests table. Although this means our time stamps can be inaccurate by 1 or 2 hours, people are not likely to be playing Dognition games at midnight, so 1-2 hours should not affect the weekdays extracted from each time stamp too much.

The functions used to subtract time differ across database systems, so you should double-check which function you need to use every time you are working with a new database. We will use the date_sub function:

https://www.w3schools.com/sql/func_mysql_date_sub.asp (https://www.w3schools.com/sql/func_mysql_date_sub.asp)

Question 11: Write a query that extracts the original created_at time stamps for rows in the complete_tests table in one column, and the created_at time stamps with 6 hours subtracted in another column. Limit your output to 100 rows.

In [18]:

```
%%sql
SELECT created_at, DATE_SUB(created_at, interval 6 hour) AS corrected_time
FROM complete_tests
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[18]:

created_at	corrected_time
2013-02-05 18:26:54	2013-02-05 12:26:54
2013-02-05 18:31:03	2013-02-05 12:31:03
2013-02-05 18:32:04	2013-02-05 12:32:04
2013-02-05 18:32:25	2013-02-05 12:32:25
2013-02-05 18:32:56	2013-02-05 12:32:56
2013-02-05 18:33:15	2013-02-05 12:33:15
2013-02-05 18:33:33	2013-02-05 12:33:33
2013-02-05 18:33:59	2013-02-05 12:33:59
2013-02-05 18:34:25	2013-02-05 12:34:25
2013-02-05 18:34:39	2013-02-05 12:34:39

Question 12: Use your query from Question 11 to adapt your query from Question 10 in order to provide a count of the number of tests completed on each day of the week, with approximate time zones taken into account, in each year in the Dognition data set. Exclude all dog_guids and user_guids with a value of "1" in their exclude columns. Sort the output by year in ascending order, and then by the total number of tests completed in descending order. HINT: Don't forget to adjust for the time zone in your DAYOFWEEK statement and your CASE statement.

In [19]:

```
%%sql
SELECT DAYOFWEEK(DATE_SUB(created_at, interval 6 hour)) AS dayasnum, YEAR(c.created_at) AS year, COUNT(c.created_
at) AS numtests,
            (CASE
             WHEN DAYOFWEEK(DATE_SUB(created_at, interval 6 hour))=1 THEN "Su"
             WHEN DAYOFWEEK(DATE_SUB(created_at, interval 6 hour))=2 THEN "Mo"
             WHEN DAYOFWEEK(DATE_SUB(created_at, interval 6 hour))=3 THEN "Tu"
             WHEN DAYOFWEEK(DATE_SUB(created_at, interval 6 hour))=4 THEN "We"
             WHEN DAYOFWEEK(DATE_SUB(created_at, interval 6 hour))=5 THEN "Th"
             WHEN DAYOFWEEK(DATE_SUB(created_at, interval 6 hour))=6 THEN "Fr"
             WHEN DAYOFWEEK(DATE_SUB(created_at, interval 6 hour))=7 THEN "Sa"
             END) AS daylabel
FROM complete_tests c JOIN
                     (SELECT DISTINCT dog_guid
                      FROM dogs d JOIN users u
                        ON d.user_guid=u.user_guid
                      WHERE ((u.exclude IS NULL OR u.exclude=0)
                      AND u.country="US"
                      AND (u.state!="HI" AND u.state!="AK")
                      AND (d.exclude IS NULL OR d.exclude=0))) AS dogs_cleaned
    ON c.dog_guid=dogs_cleaned.dog_guid
GROUP BY year,daylabel
ORDER BY year ASC, numtests DESC;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
21 rows affected.
```

Out[19]:

dayasnum	year	numtests	daylabel
1	2013	6061	Su
7	2013	4754	Sa
2	2013	3798	Mo
4	2013	3410	We
3	2013	3276	Tu
5	2013	3079	Th
6	2013	3049	Fr
2	2014	7908	Mo
1	2014	7736	Su
3	2014	6513	Tu
7	2014	6081	Sa
4	2014	5772	We
5	2014	4800	Th
6	2014	4339	Fr
1	2015	10406	Su
2	2015	8229	Mo
7	2015	7154	Sa
3	2015	6673	Tu
4	2015	6266	We
5	2015	5881	Th
6	2015	5275	Fr

You can try re-running the query with time-zone corrections of 5, 7, or 8 hours, and the results remain essentially the same. All of these analyses suggest that customers are most likely to complete tests around Sunday and Monday, and least likely to complete tests around the end of the work week, on Thursday and Friday. This is certainly valuable information for Dognition to take advantage of.

If you were presenting this information to the Dognition team, you might want to present the information in the form of a graph that you make in another program. The graph would be easier to read if the output was ordered according to the days of the week shown in standard calendars, with Monday being the first day and Sunday being the last day. MySQL provides an easy way to do this using the FIELD function in the ORDER BY statement:

https://www.virendrachandak.com/techtalk/mysql-ordering-results-by-specific-field-values/ (https://www.virendrachandak.com/techtalk/mysql-ordering-results-by-specific-field-values/)

Question 13: Adapt your query from Question 12 so that the results are sorted by year in ascending order, and then by the day of the week in the following order: Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday.

In [20]:

```
%%sql
SELECT DAYOFWEEK(DATE_SUB(created_at, interval 6 hour)) AS dayasnum, YEAR(c.created_at) AS year, COUNT(c.created_
at) AS numtests,
              (CASE
               WHEN DAYOFWEEK(DATE_SUB(created_at, interval 6 hour))=1 THEN "Su"
               WHEN DAYOFWEEK(DATE_SUB(created_at, interval 6 hour))=2 THEN "Mo"
               WHEN DAYOFWEEK(DATE_SUB(created_at, interval 6 hour))=3 THEN "Tu"
               WHEN DAYOFWEEK(DATE_SUB(created_at, interval 6 hour))=4 THEN "We"
               WHEN DAYOFWEEK(DATE_SUB(created_at, interval 6 hour))=5 THEN "Th"
               WHEN DAYOFWEEK(DATE_SUB(created_at, interval 6 hour))=6 THEN "Fr"
               WHEN DAYOFWEEK(DATE_SUB(created_at, interval 6 hour))=7 THEN "Sa"
               END) AS daylabel
FROM complete_tests c JOIN
                    (SELECT DISTINCT dog_guid
                     FROM dogs d JOIN users u
                     ON d.user_guid=u.user_guid
                     WHERE ((u.exclude IS NULL OR u.exclude=0)
                     AND u.country="US"
                     AND (u.state!="HI" AND u.state!="AK")
                     AND (d.exclude IS NULL OR d.exclude=0))) AS dogs_cleaned
    ON c.dog_guid=dogs_cleaned.dog_guid
GROUP BY year,daylabel
ORDER BY year ASC, FIELD(daylabel,'Mo','Tu','We','Th','Fr','Sa','Su');
```

```
 * mysql://studentuser:***@localhost/dognitiondb
21 rows affected.
```

Out[20]:

dayasnum	year	numtests	daylabel
2	2013	3798	Mo
3	2013	3276	Tu
4	2013	3410	We
5	2013	3079	Th
6	2013	3049	Fr
7	2013	4754	Sa
1	2013	6061	Su
2	2014	7908	Mo
3	2014	6513	Tu
4	2014	5772	We
5	2014	4800	Th
6	2014	4339	Fr
7	2014	6081	Sa
1	2014	7736	Su
2	2015	8229	Mo
3	2015	6673	Tu
4	2015	6266	We
5	2015	5881	Th
6	2015	5275	Fr
7	2015	7154	Sa
1	2015	10406	Su

Unfortunately other database platforms do not have the ORDER BY FIELD functionality. To achieve the same result in other platforms, you would have to use a CASE statement or a more advanced solution:

http://stackoverflow.com/questions/1309624/simulating-mysqls-order-by-field-in-postgresql (http://stackoverflow.com/questions/1309624/simulating-mysqls-order-by-field-in-postgresql)

The link provided above is to a discussion on stackoverflow.com. Stackoverflow is a great website that, in their words, "is a community of 4.7 million programmers, just like you, helping each other." You can ask questions about SQL queries and get help from other experts, or search through questions posted previously to see if somebody else has already asked a question that is relevant to the problem you are trying to solve. It's a great resource to use whenever you run into trouble with your queries.

2. Which states and countries have the most Dognition users?

You ended up with a pretty long and complex query in the questions above that you tested step-by-step. Many people save these types of queries so that they can be adapted for similar queries in the future without having to redesign and retest the entire query.

In the next two questions, we will practice repurposing previously-designed queries for new questions. Both questions can be answered through relatively minor modifications of the queries you wrote above.

Question 14: Which 5 states within the United States have the most Dognition customers, once all dog_guids and user_guids with a value of "1" in their exclude columns are removed? Try using the following general strategy: count how many unique user_guids are associated with dogs in the complete_tests table, break up the counts according to state, sort the results by counts of unique user_guids in descending order, and then limit your output to 5 rows. California ("CA") and New York ("NY") should be at the top of your list.

CODE

```
%%sql SELECT dogs_cleaned.state AS state, COUNT(DISTINCT dogs_cleaned.user_guid) AS numusers FROM complete_tests c JOIN (SELECT DISTINCT dog_guid, u.user_guid, u.state FROM dogs d JOIN users u ON d.user_guid=u.user_guid WHERE ((u.exclude IS NULL OR u.exclude=0) AND u.country="US" AND (d.exclude IS NULL OR d.exclude=0))) AS dogs_cleaned ON c.dog_guid=dogs_cleaned.dog_guid GROUP BY state ORDER BY numusers DESC;
```

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 56 rows affected.

In [23]:

```
%%sql
SELECT dogs_cleaned.state AS state, COUNT(DISTINCT dogs_cleaned.user_guid) AS numusers
FROM complete_tests c JOIN
                     (SELECT DISTINCT dog_guid, u.user_guid, u.state
                      FROM dogs d JOIN users u
                      ON d.user_guid=u.user_guid
                      WHERE ((u.exclude IS NULL OR u.exclude=0)
                      AND u.country="US"
                      AND (d.exclude IS NULL OR d.exclude=0))) AS dogs_cleaned
    ON c.dog_guid=dogs_cleaned.dog_guid
GROUP BY state
ORDER BY numusers DESC
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[23]:

state	numusers
CA	1363
NY	628
TX	536
FL	502
NC	467
VA	365
IL	327
PA	321
WA	294
MA	282
OH	270
CO	270
MD	239
NJ	233
GA	218
AZ	211
MI	206
CT	173
OR	153
MN	145

The number of unique Dognition users in California is more than two times greater than any other state. This information could be very helpful to Dognition. Useful follow-up questions would be: were special promotions run in California that weren't run in other states? Did Dognition use advertising channels that are particularly effective in California? If not, what traits differentiate California users from other users? Can these traits be taken advantage of in future marketing efforts or product developments?

Let's try one more analysis that examines testing circumstances from a different angle.

Question 15: Which 10 countries have the most Dognition customers, once all dog_guids and user_guids with a value of "1" in their exclude columns are removed? HINT: don't forget to remove the u.country="US" statement from your WHERE clause.

CODE

%%sql SELECT dogs_cleaned.country AS country, COUNT(DISTINCT dogs_cleaned.user_guid) AS numusers FROM complete_tests c JOIN (SELECT DISTINCT dog_guid, u.user_guid, u.country FROM dogs d JOIN users u ON d.user_guid=u.user_guid WHERE ((u.exclude IS NULL OR u.exclude=0) AND (d.exclude IS NULL OR d.exclude=0))) AS dogs_cleaned ON c.dog_guid=dogs_cleaned.dog_guid GROUP BY country ORDER BY numusers DESC;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 67 rows affected.

In [25]:

```
%%sql
SELECT dogs_cleaned.country AS country, COUNT(DISTINCT dogs_cleaned.user_guid) AS numusers
FROM complete_tests c JOIN
                   (SELECT DISTINCT dog_guid, u.user_guid, u.country
                    FROM dogs d JOIN users u
                      ON d.user_guid=u.user_guid
                    WHERE ((u.exclude IS NULL OR u.exclude=0)
                    AND (d.exclude IS NULL OR d.exclude=0))) AS dogs_cleaned
    ON c.dog_guid=dogs_cleaned.dog_guid
GROUP BY country
ORDER BY numusers DESC
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[25]:

country	numusers
US	8936
N/A	5466
CA	484
AU	142
GB	123
DE	40
NZ	38
DK	34
NO	30
FR	23

The United States, Canada, Australia, and Great Britain are the countries with the most Dognition users. N/A refers to "not applicable" which essentially means we have no usable country data from those rows. After Great Britain, the number of Dognition users drops quite a lot. This analysis suggests that Dognition is most likely to be used by English-speaking countries. One question Dognition might want to consider is whether there are any countries whose participation would dramatically increase if a translated website were available.

3. Congratulations!

You have now written many complex queries on your own that address real analysis questions about a real business problem. You know how to look up new functions, you know how to troubleshoot your queries by isolating each piece of the query until you are sure the syntax is correct, and you know where to look for help if you get stuck. You are ready to start using SQL in your own business ventures. Keep learning, keep trying new things, and keep asking questions. Congratulations for taking your career to the next level!

There is another video to watch, and of course, more exercises to work through using the Dillard's data set.

In the meantime, enjoy practicing any other queries you want to try here:

In [ ]:

```
```